

EXTERNAL LANGUAGE KLIPA FOR DIGITAL COMPUTER URAL-2

by

STAT

Marek GRONIEWSKI and Władysław TURSKI

STAT

Computation Centre of the Polish Academy of Sciences, Warsaw.

0. Introductory notes and underlying philosophy.

KLIPA /Digitalizing Translator of Address Parametres/ is an assembling and readdressing programme which replaces inscriptions by their numerical values in accordance with previously given declarations /cf. sections 4 and 5/.

A thorough formalized description of the KLIPA language is given in another paper of ours [4], thus, in the following sections of the present paper we introduce only the most important information about the formal side of the language, hoping that it will be sufficient for understanding main principles of the syntax of the language.

In the present paper the authors aimed at exhibiting such features of the KLIPA translator for the Ural-2 which are, in our opinion, most important, new, and may be useful for those who work in the field of compilers practice.

KLIPA is used for automatic performance of the following tasks:

- 0.1 Assembling of separate pieces of a programme together,
- 0.2 Selecting and including desired standard and library subroutines into main programme,
- 0.3 Assembling of sections of the main programme,
- 0.4 Calling proper sections of the programme,
- 0.5 Transfer to, and from the auxiliary storage.

Translator for KLIPA has been constructed by a team of scientists from the Computation Centre of the Polish Academy of Sciences, which included, besides the authors, Mrs. Jadwiga Empacher, Miss Jadwiga Zdanowska and Mr. Ryszard Solich.

STAT

The form of the KLIPA is by no means accidental. There are, by and large, two factors which have determined the form of KLIPA: first of all, KLIPA is an example of the authors' general view on the automatic programming and its connection

- 2 -

with the internal organisation /logical design/ of computers; secondly, the form of KLIPA is, up to certain degree, a result of unavoidable haste in which the language has been constructed.

Since we think the first factor being of the utmost importance, we shall devote additionally a few words to express our point of view.

In our opinion the logical design of most of contemporary computers is not quite fit for optimal translation from an external language into machine codes and thus the existing translators are either very quick in operation, but produce programmes very far from optimal /e.g. MAD, Manchester Autocode/, or, coming near to our understanding of the word "optimal", are hopelessly slow /e.g. SAKO/.

Some experts, noticeably Soviet ones, are of the opinion that in cases of very complicated programmes, compilers /translators/ produce programmes more optimal than those produced by a programmer working in machine codes. It seems, however, that this may be true only if one compares coding in absolute addresses with a fully automatic coding. Nevertheless, we should bear in our minds, that there exists a third coding technique, viz. quasi-manual coding which consists in preserving the form of machine codes while relying on an automatic process for performance of more tedious tasks.

We found that programming in KLIPA is both easier and quicker than it is in machine code, and there is practically nothing left to be desired better from the optimisation point of view.

Of course it should not be inferred that the authors disregard the possibilities of a real autocode. We only express the opinion, that sophisticated autocodes need computers which are build to meet very specific demands, which otherwise have to be satisfied by long and time-consuming sequences of operations.

And the Ural-2 computer is anything but such a computer. When constructing the KLIPA the authors have found very useful the experience gained during exploitation of the Emal-2, a Polish made drum computer, operated in the Centre for two years.

1. Description of the Ural-2

In the table 1.1 main characteristics of the computer are listed. The tape reader was attached to the Ural-2 in our Centre in order to enable the work with an alphanumerical code, which was impossible on the original, standard Ural-2 equipment.

2. Set of employed symbols.

KLIPA uses the Second International Teleprinter Code which allows for using of:

<letters> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
 <figures> ::= 1|2|3|4|5|6|7|8|9|0|
 <delimiters> ::= +|-|=|:|.|'|$\frac{\quad}{\quad}$|SP|LF|CR|LS|FS|Blank

3. Inscriptions.

KLIPA uses a fixed set of inscriptions which may consist of one or more letters, e.g. x, sin, delta, april. Inscriptions are used for representing: directives, address parametres, indices, variable and procedure /subroutine/ identifiers. Variable identifiers may depend on parametres, which may be either symbolic /running indices/ or fixed. If they are fixed, they may assume one the following forms, which are identical:

$$\text{delta } (5) = \text{delta } + 5 = \text{delta } 5$$

In the case of symbolic indices, the most elaborate form they may assume is: $\text{c}n + \beta$, where c and β stand for any octal numbers /not exceeding, however, 7777/ and n is any variable identifier; thus it is possible to write in correct KLIPA statement:

$$\text{kappa } (15\text{pi} + 637).$$

which is understood by the translator as 15pi + 637th variable in an array of kappas.

4. Searching.

In KLIPA translator a new method for identification of inscriptions has been employed. This method termed "characteristic function method" is, in a sense, a development of ideas presented in [2] and [3], and can be successfully used in all procedures in which it is necessary to establish a correspondence between alphanumerical words and numerical values, e.g. in automatic translation.

- 5 -

if ξ is an positive integer, $\phi(\xi)$ should also be a positive integer.

In the most obvious application ϕ is the address in which information about the word ξ is stored.

The simplest transform, viz. the linear one: $y = ax+c$, is quite good for our purposes, especially if ^{for} various segments $\xi_j \in \xi_{j+1}$ we choose various constants a_j and c_j . One should always remember, that ϕ must be one-valued transform, i.e.

$$\phi(\xi_j) \neq \phi(\xi_j) \text{ if } \xi_1 \neq \xi_j.$$

Fig. 4.1 presents the geometrical interpretation of the ideas explained above, for a vocabulary consisting of two one-letter words, and four two-letter words, and for a quinary code.

In table 4.2 words, their quinary representation, appropriate transforms and address directory for this example are listed

Table 4.2

Word	Quinary representation	transform	address
□	1	$y = \frac{1}{2}x + \frac{1}{2}$	1
△	3		2
□△	13	$y = \frac{1}{3}x + 2$	3
○△	24		5
△○	32		6
△△	43		8

Wasted registers are 4 and 7

In practice, the KLIPA translator proceeds along slightly different lines. Consider first n letters of a given inscription. Letters $n+1$ st, $n+2$ nd, ... are skipped out. If an inscription contains less than n letters, than the following procedure is interrupted after the last letter is being read. As soon as the j -th letter /treated as j -th "32-basis-system" digit x_j / is being read, the computer evaluates:

$$\hat{f}_j := a_j f_{j-1} + x_j + c_j, \text{ if } \hat{f}_j \geq M \text{ then}$$

$$f_j := \hat{f}_j - K, \text{ otherwise } f_j := \hat{f}_j.$$

For a set of inscriptions which is used by KLIPA language, following values of constantes have been found to satisfy the optimisation demands:

- 6 -

$$n=4, M=255, N=179, a_1=2 \text{ for } i=2,3,4,$$

$$c_1=0, c_2=20, c_3=-58, c_4=25, f_0=0.$$

Using such a transform we got:

$$\sum_{j=0}^4 (1 - c_j(y)) = 0.2$$

Such a loss of memory cells is amply rewarded by a very high speed of translation achieved due to quick identification of inscriptions occurring in the programmes. As a matter of fact KLIPA programmes are translated at the speed allowed by the tape reader, i.e. about 400 characters per second.

A method for determining the coefficients of the transform described above is now being developed by the authors and will be published separately.

5. Allocation of storage space.

A programme written in KLIPA is divided into sections. When the programme is being obeyed one section only is recorded in the main store. Sections may be interchanged and behave in the way exactly similar to chapters in Manchester Autocode [1]. There is one significant difference, however, namely, the section length, though being fixed for a given programme, may vary in certain limits from one programme to another, accordingly to the programmer's desire. A skilful programmer may derive obvious advantages from this feature of KLIPA.

Since such a feature of an external language is somewhat unusual, we take the liberty to give some details about how it is done. During execution of a programme written in KLIPA, in the main store there are some registers allocated for working cells of standard subroutines and for parts of translator/red-tape routines of it/. The remainder of the store constitutes a "programmer's little acre", which he should divide between variables /and constants/ of the programme, and the programme /or rather a section of it/ itself. Numerical quantities may be regarded as forming two classes. All the quantities, which are needed for this section of the programme which is actually being obeyed, and only for this section belong to the first class. All quantities needed in at least

two sections belong to the second class, and will be henceforth called general variables. It is important the general variables should be "lost" when sections are interchanged. The directive zostaw: N, where N is any previously declared parameter or octal number, precedes the body of a programme, and results in initiating special red-tape subroutine of translator which divides the remainder of the store between the mentioned parts. Needless to say, the number /or parameter/ N indicates how many registers are to be reserved for general variables. The same routine evaluates automatically the smallest possible label that may be used in sections of this particular programme which was preceded by a zostaw: N directive. This smallest value is afterwards automatically inserted everywhere instead of pc /short for "początek"= beginning/ labels. The pc provides a convenient "zero point" for relative addressing.

6. Instruction form.

A KLIPA instruction consists of two parts:

- a. operational part,
- b. address part.

The first part consists of two octal digits which may be preceded by a - sign, if the instruction is to be modified in side of an automatic cycle.

The address part is separated from operational part by any number of SP symbols /spaces/ and may be written in any of the following forms: an octal number, an inscription, and an octal number, an inscription and an octal number separated by the - or + sign, an inscription with a parameter embraced in round brackets. All octal numbers in the address part must not exceed 7777. The address part may as well be empty.

Examples of KLIPA instructions:

37		
41	176	
-56	beta	note: klucz = the key
23	klucz 4	
22	k + 11	
27	gemmas - 7654	
-02	a, (2n - 5)	

7. Calling in subroutines.

The standard subroutine list for KLIPA includes: sin, cos, tan, arctan, exp, log, sh, ch, th, entier, rq /square root/, rc /cubic root/, czytaj /for reading and converting into binaries one decimal number/, drukuj /for printing out one decimal number/ and some auxiliary subroutines. Calling is best explained by examples:

a.	42	x	b.	42	x
	22	sin		22	sin
	56	y		22	rq
				56	y

In the example a. $y = \sin(x)$, and in the example b. $y = \sqrt{\sin(x)}$ are evaluated.

Similarly to the Manchester Autocode any of the standard functions may be treated as a "quandle" /cf. [1]/, if desired.

A programmer does not declare the values of inscriptions used for identification of standard procedures /subroutines/.

8. Modification.

One more feature of the KLIPA translator deserves to be mentioned. In the translator we very often use the machine instruction 30 α . This instruction allows for using any main store register as an "B register". If any other instruction is preceded by the 30 α , the computer adds the content of α register to that instruction and obeys such a compound instruction. This is frequently used for organising multiway switches and checking the formal correctness of a programme written in KLIPA.

A teleprinter symbol has meaning depending on the position in the programme in which it occurs. e.g. a $\frac{1}{2}$ sign after directive blok denotes the end of the block of numbers, in all other positions the $\frac{1}{2}$ must be interpreted as a formal error. During translation of a programme, in the main store there is so-called dictionary of symbols, containing 32 items, i.e. one item for each teleprinter symbol. This dictionary occupies locations from k to $k + 31$ inclusively. As soon as the directive blok is identified on the $k + 22$ nd cell a jump instruction is sent. This jump instruction leads to another

References

- [1] R.A. Brooks.: *Flowchart Antecode: Principles of the Programming Library, Annual Review in Automatic Programming, Vol.I, p.93, Pergamon Press 1960.*
- [2] M. Greniewski.: *Wstęp do Programowania i Modelowania Cyfrowego. /An Introduction to Programming and Digital Simulating/ PWN, Warsaw, 1961.*
- [3] M. Greniewski.: *Synthetic Modifiers' Code /for the Emal-2 Digital Computer/ Warsaw, 1961.*
- [4] M. Greniewski und W. Turaki.: *Beschreibung der Sprache KLIPA, Proceeding of the Dresden Symposium on Automatic Computing Machines. /In print/.*

- 11 -

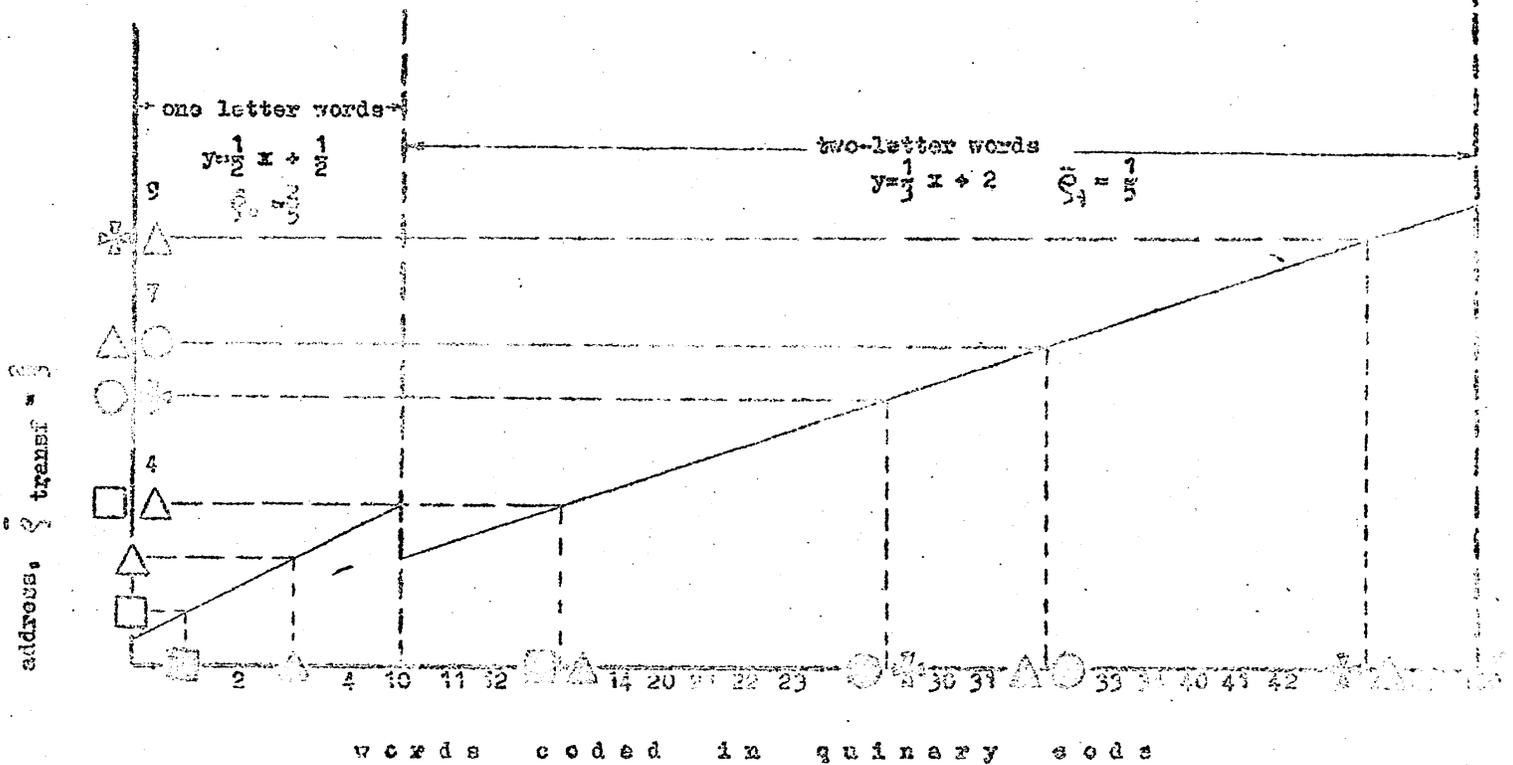
1.1

Main characteristics of the Ural computer

Arithmetics	Binary, modulus - sign
Point	, Fixed and floating
Instruction length	20 bit
Floating point number	40 bit
Fixed point number	20 bit
Main store	Ferrite cores for 4096 words of 20 bit or 2048 words of 40 bit
Backing store	Two magnetic drums for 8192 words of 40 bit each, and one unit of magnetic tape which can keep up to 256 zones, each up to 4096 words of 20 bit
Speed	10,000 - 12,000 operations per second in fixed point, and half of it in float- ing point.
Input	Ferranti tape reader TR3B
Output	Numerical line printer of 16 characters /printing one number to a line at the rate of 25 lines per second/.

Fig. 4.1

Simple transform of numerical representation of coded words: $y_j = a_j x_j + c_j$ allows for very easy and quick retrieval of information.



*) Deformation of spacing on x-scale due to graphical reasons.